STOCK BROKERAGE APPLICATION

PROJECT REPORT

GROUP Members: **TEAM BIRYANI**

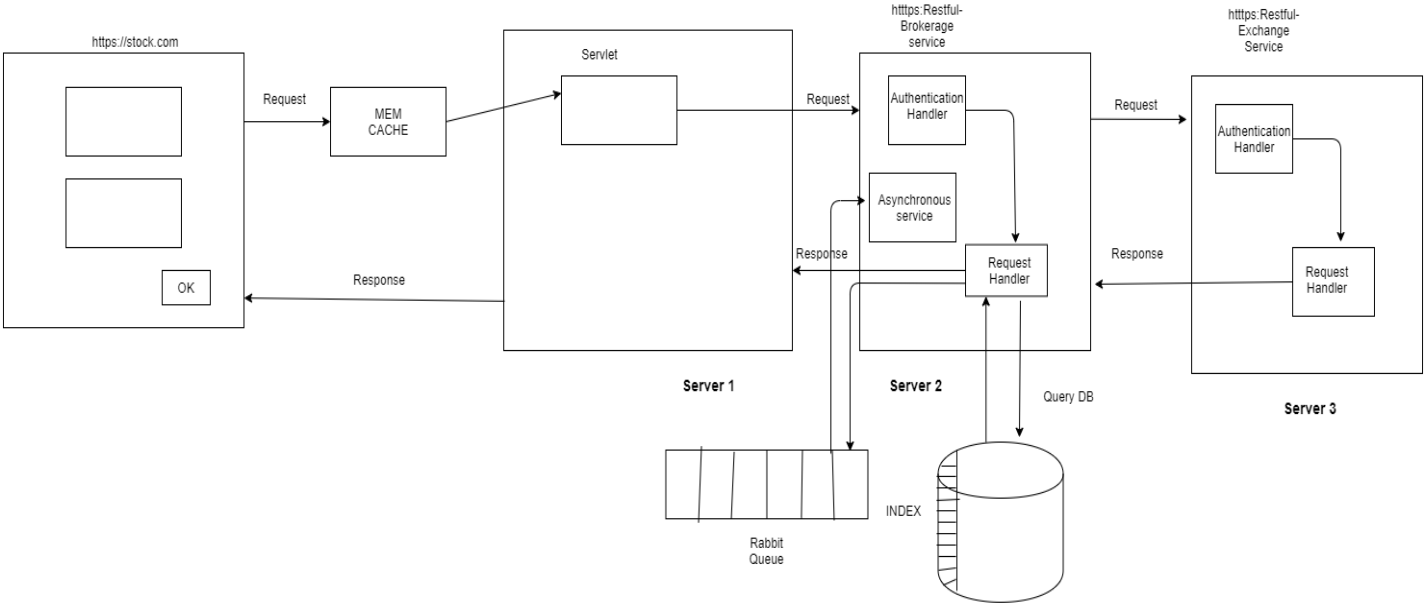
AMTUL NAZNEEN (axn180041)

HEMANJENI KUNDEM (hxk180032)

ASMA SADAT (AXS180236)

Table of Contents

Architecture



Architecture Explanation:

Server1 handles the servlets responsible for JSPs .

Server2 hosts the webservices ,business logic and manages the database connections.

Server3 hosts the business data for the application.

Each time a request is sent from the server1 to server 2, Memcache checks for the record requested. If it has record for the query, then it returns it. Else the request gets forwarded to server2 on HTTPs and inserted in cache for future queries.

If the webservice recurring buy/sell has to be called, the servlet puts the request in the Rabbit MQ at the server 2 from where the server 2 program continuously reads the incoming requests. For all other functionalities no MQ is used. All the incoming requests which include user operations like buy/sell stocks, add accounts, transfer money etc., are handled by server 2 .This server makes appropriate changes to DB ,simultaneously forwarding requests for fetching current prices and past stock prices to the 3rd server. The 3rd server returns the price data to the 2nd server which in turn returns to user. The server1 cannot contact the server 3 directly. All requests have to route through server2.

Project Structure

1. **stock-brokerage-website** – Code for the servlets, JSP and Server(server1) running for the servlets cache management .

2. **stock-brokerage-webapp** -- Code for JAVA REST web services(JAX-RS), scripts for DB creation and insertion of sample data, Server for handling HTTPS requests , queue management for reading recurring buy/sell requests from the stock-brokerage-website server(server 2).

3. **stock-exhange-webapp** – Code for returning static data for buy and sell functionalities , and generating dynamic current stock prices. Server(server 3) listening to HTTPS requests for handling these requests.JAVA REST webservices(JAX-RS) for returning the HTTPS calls .

Technologies Used

UI : JSP with servlets, CSS, Bootstrap, JavaScript, jQuery,HTML5

We added Bootstrap for faster UI development using its rich library set.

Backend: Java

All the team members were familiar with java . Hence we chose Java for backend implementation.

Database: MySQL

MySQL database is used as it is freeware and comfortable to use. Oracle DB provided by the department is prone to connection issues.

Caching : Memcached

For caching at the server 1 side, used Memcached service. This handles the requests from servlets for list of companies and current prices of companies. We preferred Memcached cache as it was free and widely used.

Queuing : RabbitMQ

RabbitMQ resides at server 2 side, This handles the requests generated from server 1 for recurring sell and recurring buy. Implementation of RabbitMQ is easy and less time consuming.

Compression: Native Tomcat support

Requests/responses gets exchanged in 'gzip' encoding between servers, thanks to this feature.

Security: TLS/SSL

All servers accept only secure connections, and if any connection is attempted using a http protocol, services are not called.

Authentication between servers: Token-based. The implementation has a single token hardcoded at all the servers. So if any other outside(untrusted) service makes a request, error message is returned. Even though a user is authenticated, server 1 should possess this token for the webservices to respond.

Webservices: JAX-RS

Implementation using JAX-RS is straight forward, annotations are provided for easier implementation. All webservices return XML data to the calling server.

Web Services Used

verifyLoginUser

- An existing user should login through his account by providing username and password. This service checks if the password, user credentials match with the database and grants access.
- Input – form data from POST method called by login.jsp handled by Login.java,
- Output – XML in the following format:

```
<USER>
   <USERNAME>darkpheonix</USERNAME>
   <EMAIL>jean.grey@xmen.com</EMAIL>
  <ADDRESS>New York, New York</ADDRESS>
</USER>
```

allCompanies

- Gives a list of companies to buy/sell stocks from.
- Input - form data from POST method called by SearchStocks.jsp handled by PopulateCompany.java,
- Output – XML in the following format

```
<COMPANIES>
   <COMPANY name=”Facebook“>
</COMPANIES>
```

signUpUser

- Creates a new user in the system taking into input the signup details from servlets.
- Input - form data from POST method called by Registration.jsp handled by Registration.java
- Output – XML in the following format

```
<USER>
   <USERNAME>darkpheonix</USERNAME>
   <EMAIL>jean.grey@xmen.com</EMAIL>
  <ADDRESS>New York, New York</ADDRESS>
</USER>
```

forgotPwd

- Enables user to reset an existing password. An email is sent to < qwertyuioplkjhg49@outlook.com > with an OTP.
- Input - form data from POST method called by ForgotPassword.jsp handled by ForgotPassword.java,
- Output – XML in the following format

```
<RESET>
        <STATUS>true/false</STATUS>
</RESET>
```

resetUserPwd

- Enables the user to set a new password after gaining access with temporary OTP.
- Enables user to reset an existing password. An email is sent to < qwertyuioplkjhg49@outlook.com > with an OTP.
- Input - form data from POST method called by ResetPassword.jsp and handled by ResetPassword.java.
- Output – XML in the following format

```
<RESET>
        <STATUS>true/false</STATUS>
</RESET>
```

recoverAccount

- Enables the user to enter an email address to recover his account and redirects to profile page.
- Input - form data from POST method taking username,password is called by RecoverAccount.jsp and handled by RecoverAccount.java
- Output- XML in the following format

```
<USER>
      <USERNAME>darkpheonix</USERNAME>
      <EMAIL>jean.grey@xmen.com</EMAIL>
     <ADDRESS>New York, New York</ADDRESS>
</USER>
```

userSchedule

- Enables user to view all his recurring buy/sell stock schedules.
- Input - form data from POST method called by MySchedules.jsp and handled by MySchedules.java
- Output- XML in the following format

```
<SCHEDULE>
      <STOCK_ID>id </STOCK_ID>
      <COMPANY_NAME> Facebook</ COMPANY_NAME >
      < QUANTITY>qty</ QUANTITY>
      <PRICE>30</PRICE>
      < SCHEDULED_DATE>12-12-2019</ SCHEDULED_DATE>
      < RECURSION_PLAN>WEEKLY</ RECURSION_PLAN>
      <PURCHASE_STATUS>PURCHASED</ PURCHASE_STATUS>
</SCHEDULE>
```

userStocks

- Returns the stocks owned by a user
- Input - form data from POST method called by Mystocks.jsp and handled by Mystocks.java
- Output- XML in the following format

```
<STOCKS>
        <STOCK>
            <PRICE>2370.09</PRICE>
```

```
            <DOP>2019-12-02</DOP>
            <QUANTITY>2</QUANTITY>
            <COMPANY>Facebook</COMPANY>
            <STOCKID>2</STOCKID>
        </STOCK>
</STOCKS>
```

userAccounts

- Returns the bank accounts of a user along with routing information.
- Input - form data from POST method called by ViewAccounts.jsp and handled by ViewAccounts.java
- Output- XML in the following format:

```
<BANK_ACCOUNTS>
    <ACCOUNT>
        <ACCOUNT_NO>012098323</ACCOUNT_NO>
        <ROUTING_NO>11241434</ROUTING_NO>
        <BALANCE>100.90</BALANCE>
    </ACCOUNT>
</BANK_ACCOUNTS>
```

addAccount

- Enables a user to add a bank account
- Input - form data from POST method called by AddAccount.jsp and handled by AddAccount.java
- Output- XML in the following format

```
<ACCOUNT>
    <STATUS>true/false</STATUS>
</ACCOUNT>
```

transferAmount

- Enable users to transfer amounts from one account to another.
- Input - form data from POST method called by TransferAmount.jsp and handled by TransferAmount.java
- Output- XML in the following format:

```
<TRANSFER>
    <STATUS>true/false</STATUS>
</TRANSFER>
```

deleteSchedule

- Enables user to stop a recurring buy.
- Input - form data from POST method called by DeleteSchedule.jsp and handled by DeleteSchedule.java
- Output- XML in the following format:

```
<DELETE_SCHEDULE>
    <STATUS>true/false</STATUS>
</DELETE_SCHEDULE>
```

deleteSellSchedule

- Enables user to stop a recurring sell.
- Input - form data from POST method called by MySellSchedules.jsp and handled by DeleteSellSchedule.java
- Output- XML in the following format:

```
<DELETE_SCHEDULE>
    <STATUS>true/false</STATUS>
</DELETE_SCHEDULE>
```

editProfile

- Enables user to change his profile details
- Input - form data from POST method called by EditProfile.jsp and handled by EditProfile.java
- Output- redirect to userProfile.java

buyStocks

- Enables user to perform multiple buys and also schedule recurring buys for a given stock
- Input - form data from POST method called by PurchaseOptions.jsp and handled by BuyStocks.java
- Output- XML in the following format:

```
<PURCHASE>
    <STATUS>true/false </STATUS>
</PURCHASE>
```

sellStocks

- Enables user to perform multiple sells and also schedule recurring sells for a given stock.
- Input - form data from POST method called by Buy.jsp and handled by SellStocks.java
- Output- XML in the following format:

```
<SELL>
    <STATUS>true/false </STATUS>
</SELL>
```

userSellSchedule

- Returns all the recurring transactions user scheduled to sell.
- Input - form data from POST method called by Myschedule.jsp and handled by MySchedule.java
- Output- XML in the following format:

```
<SCHEDULES>
        <SCHEDULE>
            <PURCHASE_STATUS>Sold</PURCHASE_STATUS>
```

```
            <RECURSION_PLAN>Weekly</RECURSION_PLAN>
            <RECURSION_PLAN>2019-12-02</RECURSION_PLAN>
            <QUANTITY>2</QUANTITY>
            <COMPANY_NAME>Facebook</COMPANY_NAME>
            <STOCKID>2</STOCKID>
        </SCHEDULE>
</SCHEDULES>
```

companyPrice (Server1 & 2)

- Returns the current price of a company given by the user and also the past prices of the company stocks (upto 5 years) and sends this information to stock-brokerage-webapp server.
- Input - form data from POST method called by BuyOptions.jsp and handled by BuyOptions.java
- Output- XML in the following format:

```
<STOCKS>
   <STOCK date=”2019-12-02” price=”9087.09/>    .
   <STOCK date=”2019-12-03” price=”98.23/>    .
</STOCKS>
```

Website Design and Functionalities Offered

- The main page of the user shows accounts available, transfers, my schedules, my stocks which on clicking direct the user to the corresponding pages.
- My schedules displays a table which lists user stocks.
- User can search for companies of which he wants to buy stocks and see the real time price of that company’s stock.
- User can check the stock selling history of a company by current day, current week, past week, month-to-date, year-to-date and past 5 years.
- He can buy or sell multiple stocks of a particular company.
- User can edit his details by going to his profile page.
- User can make it automatic by selecting recurring option. Where stock is either sold or bought every week, monthly or annually.
- User can transfer amount to another user using the bank account number and routing number.
- Any number of bank accounts can be added and deleted to the account when we click on my accounts in the nav bar.
- User can reset his password by providing his username and email address. If he forgets his password then a code is sent to his email which on entering at the recover account pe will recover it.

Problems Encountered in our Project

1. Setting up TLS/SSL, specifically in extracting the self-signed certificates and placing them in the relevant server's configuration files
2. Determination of object serialization that is needed during Caching and Queuing the necessary objects
3. Migration of database from UTD Oracle SQL to MySQL because of a 2 day long downtime experienced with CS department servers